Toward a Taxonomy of LLM Roles and Interaction Patterns in Game Content Generation Systems

Ashley Hart
Department of Computer and Information Science and Engineering
University of Florida
Gainesville, FL, USA
ashley.hart@ufl.edu

Abstract

Recent work at the intersection of procedural content generation (PCG) and large language models (LLMs) has given rise to a body of research where LLMs are embedded within, utilized as, or interacting with procedural content generation pipelines. The term PCG-LLM system is operationalized to distinguish systems that contain an LLM and another separate and distinct component that utilizes traditional PCG approaches to influence the final content from systems where this separation is not as clear. Additionally, despite increasing adoption, there is a need to clarify and standardize terms for describing LLM roles and interaction patterns with other components within game content generation systems. This work analyzes existing systems, examines their eligibility as PCG-LLM systems, and proposes categorizations of LLM roles and interaction patterns observed within them. By providing a common vocabulary for characterizing how LLMs are used within PCG pipelines, this proposed taxonomy aims to reduce ambiguity, support clearer comparison between approaches, and encourage more systematic design and evaluation. The categorizations presented here highlight underexplored areas and invite researchers and practitioners to adopt, critique, and extend this taxonomy as the field continues to evolve.

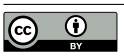
Keywords

Procedural content generation, large language models, taxonomy, software architecture, game content

ACM Reference Format:
Ashley Hart. 2026. Toward a Taxonomy of LLM Roles and Interaction Patterns in Game Content Generation Systems. In *Foundations of Digital Games (FDG '26), August 10–13, 2026, Copenhagen, Denmark.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3815598.3815694

1 Introduction

Procedural content generation (PCG) is the algorithmic creation of game content [25], however, with the rise of generative AI and the application of LLMs within the games sector, it is increasingly harder to determine where PCG ends and generative AI begins. Some may argue that LLMs are a part of the algorithmic process, whereas others may argue that LLMs are black boxes that should not be considered a part of an algorithmic design process.



FDG '26, Copenhagen, Denmark

ACM ISBN 979-8-4007-2495-4/26/08
https://doi.org/10.1145/3815598.3815694

The boundary between PCG and other generative approaches became increasingly blurred following the release of GPT-2 in 2019 [19], which demonstrated the ability to generalize to unseen tasks. The capacity for generative AI models to generalize catalyzed a slew of research across many fields, including procedural content generation.

Since then, researchers have increasingly applied LLMs to PCG tasks including level generation, narrative creation, and complete game design. A prominent early example is MarioGPT [22] which was published in 2023, utilizing an LLM and a frozen text encoder to generate *Super Mario Bros* [16] levels. However, these approaches are rather diverse and domain dependent, making comparisons between systems difficult. This variance supported the strength of LLMs generalization capabilities, but also gave rise to terminology overload when trying to describe how the LLM was integrated and what it contributed to the generation process.

As such, there is a need for standardized terminology that describes a system that utilizes both PCG and LLMs within its content creation process, the roles LLMs can play within such systems, and the interaction patterns between LLM-powered components and others. All the surveyed applications use an LLM for PCG, but an LLM is not always replacing the algorithmic methods that defined earlier PCG approaches. This further demonstrates the need for terms that distinguish how an LLM is being utilized within a generation pipeline. A review of past literature shows that researchers resort to ad-hoc descriptions of their pipelines, which introduces variance that can hinder comparative analyses across systems. This lack also hinders research in explainability, through the absence of a common ground between the systems. Establishing a shared terminology is essential for deeper architectural inquiry of game content generation pipelines.

Furthermore, there is more room to explore how fundamental architectural differences within a system influence the quality of generated content, evaluation methods, and explainability attempts. What roles can LLMs assume within a game content generation task? How do LLMs interact with traditional PCG components vs components steered by other generative models? What are the implications of an LLM's role on the resultant content? What are the implications of these roles and interaction patterns?

Without systematic categorization, the field cannot effectively compare approaches, identify best practices, or guide practitioners in architectural decisions. This work operationalizes the term PCG-LLM system and proposes categorizations for LLM roles and interaction patterns within a PCG pipeline.

From these observations came the following research questions which this taxonomy aims to answer:

- **RQ1:** What functional roles do LLMs assume in game content generation systems?
- **RQ2:** What architectural patterns exist between LLMs and other system components (where present) within a game content generation system?
- **RQ3:** What design and evaluation implications emerge from the architectural patterns found in game content generation systems?

Regarding scope, this work does not attempt to resolve the debate between what is or what is not PCG, as past works have attempted to do [25]. Additionally, this work does not consider applications of LLMs in content generation for anything beyond game content. Instead, this work considers past literature and makes the following contributions:

(1) Operationalization of the term PCG-LLM system to describe a class of systems that utilize distinct PCG and LLM components in game content generation procedures.
(2) Proposed terminology for describing the roles an LLM can assume within a game content generation system.
(3) Proposed terminology for interaction patterns between LLMs and other components (where applicable) within a PCG-LLM system, supporting comparative analyses between generation pipelines.
(4) Identification and discussion of design and evaluation implications, and the benefits of this work for the field.

The categories proposed in this work are not intended as a final classification scheme, but as a foundation for community discussion and a step toward the cultivation of shared reporting standards for software artifacts at the intersection of PCG and machine learning. The author welcomes critique, extension, and refinement as the field continues to mature.

2 Related Work

The broad applicability of PCG and LLMs have inspired other game researchers to propose terminology and groupings to support further exploration.

Taxonomies within PCG and games research are not new. Togelius et al. [26] proposed a foundational taxonomy for search-based PCG, organizing approaches by the type of content generated, how the content is represented, and how its quality is assessed. The subjectivity inherent to PCG applications [25] has long motivated researchers to develop evaluation frameworks such as the PCG Benchmark by Khalifa et al. [6] for comparing approaches across different games and evaluation metrics.

The notion of using Procedural Content Generation via Machine Learning was formalized by Summerville et al., [23] and distinguished itself from prior approaches, such as search-based PCG and experience-driven PCG, by emphasizing the model as the creator of the game content, and the generation of process as a traversal of a content space. Approaches within this categorization depended on the training data available, the representation of the desired content when interfacing with the model, and the training approach that was utilized. PCGML taxonomizes applications in this space with respect to the data's representation and the training method utilized. Liu et al. [11] built upon PCGML with their survey on how PCG is influenced by deep learning, capturing the applications, potential and constraints with utilizing these approaches. PCGDL methods are a subset of PCGML methods, and as such, the use of LLMs for PCG is a subset of both PCGML and PCGDL. Large language models are a natural extension of these works, as they are specialized deep learning models that have been applied to content generation tasks for games.

Mao et al. [13] survey the application of generative AI to PCG in a broader context, covering approaches including GANs, diffusion models, and transformers — situating LLM-based approaches as one thread within a wider generative AI movement in the field.

Gallotta et al. [3] surveyed the state of the art for applying LLMs to the games space. They defined roles an LLM could assume within or in support of a game (e.g. Player, Design Assistant, Game Master) to improve experiences for players, developers and spectators. These roles posited by Gallotta et al. operate at a conceptual level tethered to the game itself and the experiences it produces. In contrast, the terminology introduced in this work is distinct: rather than characterizing what an LLM does *for* a game, this work characterizes what an LLM does *within* a system's software architecture.

Maleki and Zhao [12] survey different PCG approaches with emphasis on works that incorporate LLMs into application spaces typically addressed by PCG. Their work taxonomizes PCG methods and game content types, highlighting how LLMs contribute to a combinatorial trend of merging generation approaches, and note limitations in transparency and evaluation that arise from reliance on closed-sourced models in past works. Their survey identifies several systems that incorporate LLMs into content generation tasks, but they do not examine the software architecture of those systems in depth.

This work sits orthogonal to these surveys, and addresses researchers' need for a common ground closer to the code. This taxonomy provides a shared language for researchers designing, evaluating and comparing systems at this intersection. The author welcomes contributions and revisions of this taxonomy as the space matures.

3 Methodology

Relevant literature was collected via foundational PCG texts [21, 27], citation tracing, targeted keyword searches, and paper suggestions from others within the field across Google Scholar, arXiv, IEEE Xplore, and the ACM Digital Library. Nine papers published between 2023 and 2024 were identified as a representative sample of systems that either incorporated PCG and LLMs in a content generation task, or utilized LLMs in applications historically addressed by PCG.

Analysis of these nine papers with respect to their functionality and structural characteristics identified two axes on which these systems vary: the role the LLM assumes, and the interaction patterns that LLM components have with others within a PCG pipeline. This motivates a two-dimensional taxonomy.

The large variety of LLM roles in surveyed works often makes categorization ambiguous. To account for this, the taxonomy employs a hierarchical ranking of LLM roles for each system, allowing for primary and secondary roles, with room for more if needed. Roles ranked higher indicate that LLMs functioning within those roles had greater impact on the final content. Interaction patterns

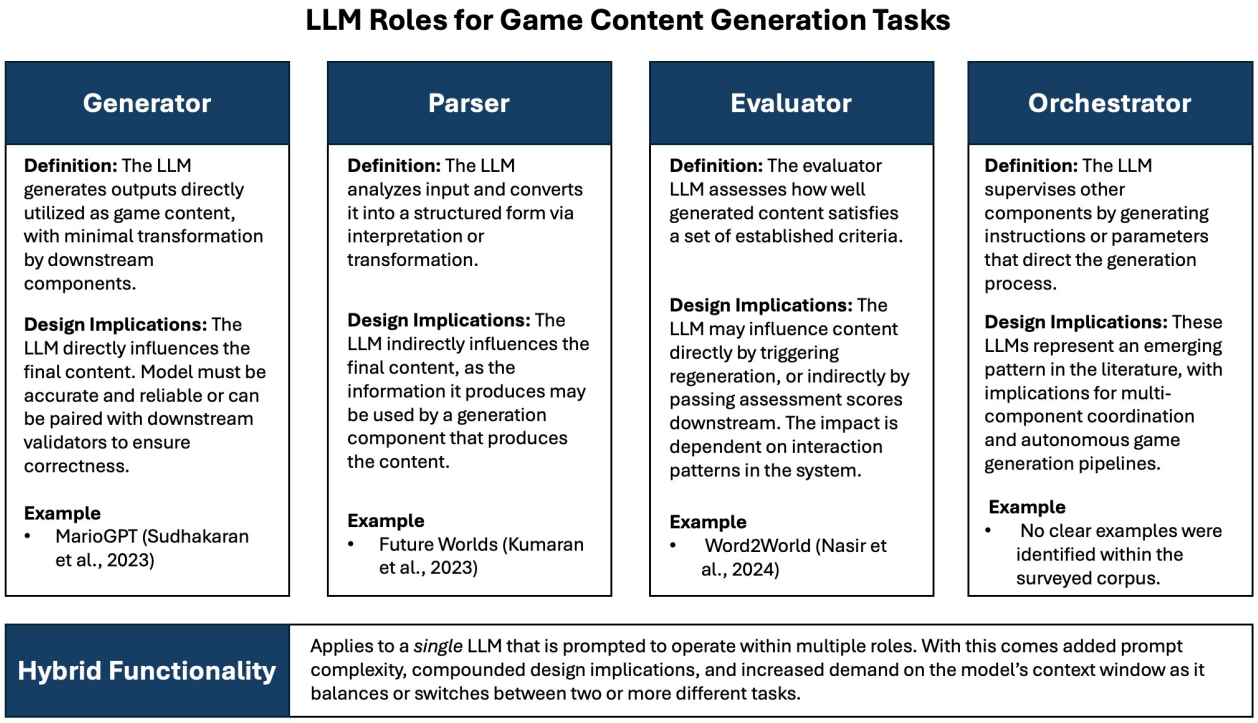


Figure 1: Four functional roles an LLM can assume in a game content generation system, presented with design implications and examples from the surveyed corpus. Hybrid functionality is presented as a model that assumes two or more roles during the generation process.

were labeled based on their presence within each work. Papers that included system flowcharts aided classification; for those that did not, interaction patterns were inferred from system descriptions and figures.

Where deemed necessary, category definitions were updated or a limitation of the taxonomy was noted. The resultant classifications can be seen in Table 1 and Table 2.

4 Taxonomy

This section presents the taxonomy and guidance on what qualifies a system for each of the categories within it. This section also covers the definition of a software component and the idea of directness, which are foundational to the correct application of this work. This taxonomy covers PCG-LLM system eligibility criteria, introduces roles the LLMs can assume within a game content generator, and interaction patterns that can exist between an LLM and other components. The design implications of each category are also discussed.

Note that the taxonomy terms proposed in this work — roles and interaction patterns — can be applied to any system that incorporates an LLM into a game content generation process, regardless of whether that system qualifies as a PCG-LLM system. Table 1 addresses PCG-LLM system eligibility, while Table 2 applies the taxonomy's vocabulary to all surveyed systems. This label is intended to distinguish systems where PCG and LLM components are separate and distinct, and not to qualify or disqualify systems from the roles and interaction patterns that follow.

4.1 Components and Directness

Making classifications with this taxonomy requires definitions for a system component, and the concept of directness. Within a software architecture, a *component* is a functional unit within a system that serves one purpose. For example, in MarioGPT [22], the natural language input is processed by a frozen text encoder. A frozen text encoder is considered to be a single component. In other words, a component transforms an input into an output and is defined by its purpose of transforming data within the pipeline. Additionally, a single LLM could also be a component. The model receives an input. The model executes a task. The model produces an output. Note that the definition of a component is not concerned with the type of input or output. In the eyes of this taxonomy, component is meant to be a term that can be applied to different types of components operating in different capacities and contexts.

Another idea to consider when classifying system components with this taxonomy is the notion of *directness*. Directness describes how much the LLM's outputs influence the final content. An LLM whose outputs are transformed by downstream components before reaching the player exerts indirect influence — for example, an LLM extracting narrative themes that prime a background generator. An LLM whose outputs are passed directly to a renderer without

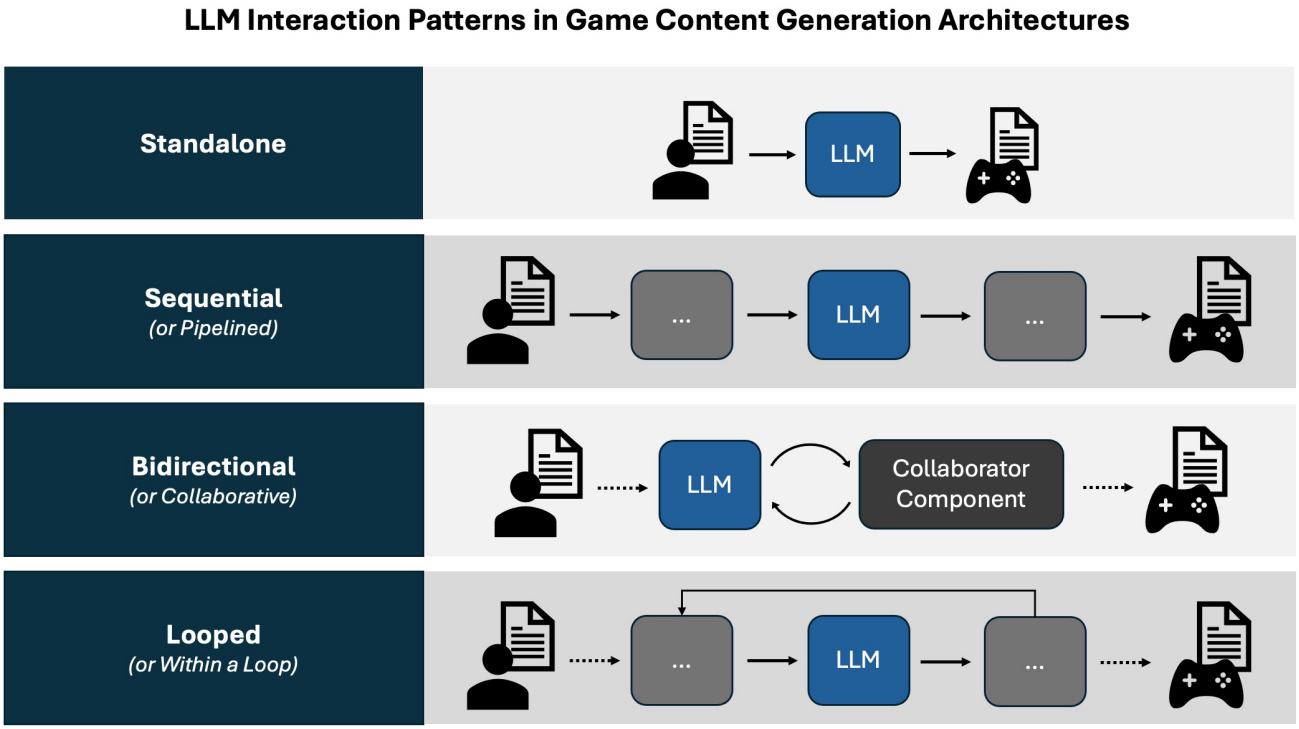


Figure 2: Four interaction patterns describing how LLM components may participate within a game content generation pipeline. Bidirectional and looped patterns are illustrated with dashed input and output arrows, as the entry and exit points for each system will vary based on the designer's implementation.

meaningful transformation exerts direct influence — for example, an LLM producing an ASCII tile map that a renderer displays in-game.

4.2 PCG-LLM Systems

A *PCG-LLM system* is a game content generation system with at least one each of an LLM — whether that LLM is generating, parsing, or evaluating content — and a traditional PCG component, both of which contribute to the final content such that removing either would fundamentally alter the output.

For example, a noise-based terrain generator that uses Perlin noise [18] and a renderer to create maps for a game is *not* a PCG-LLM system, due to the lack of an LLM. A noise-based terrain generator that receives natural language input from a user, parses it with an LLM, and uses the LLM's outputs as a set of parameters for the terrain generation *is* a PCG-LLM system. This is because the LLM prepares parameters for the PCG system components, which act independently to produce the final content. An LLM that is prompted to generate a text-based 2D map for a game is *not* a PCG-LLM system if its outputs are not further manipulated by other downstream PCG components.

Table 1 presents a classification of surveyed works and justifications for whether or not they are PCG-LLM systems. LLMaker [2] is an edge case due to a lack of architectural details about the system, indicating that accurate classification is highly dependent on available system information. The precise boundaries of this definition are left open to refinement by the community as the space matures. The author acknowledges that using separability between PCG and LLM components as a distinguishing criterion may invite debate, and welcomes that discussion as part of the taxonomy's ongoing development.

4.3 LLM Functional Roles Within a PCG-LLM System

The versatility of LLMs calls for roles to be defined so different applications of these models can be classified for system comparisons and evaluations. This taxonomy proposes four functional roles that an LLM may assume within a PCG-LLM system: Generator, Parser, Evaluator, or Orchestrator. The author also encourages the use of *LLM-as-generator*, *LLM-as-parser*, *LLM-as-evaluator*, and *LLM-as-orchestrator* when discussing LLMs fulfilling these roles. This work also discusses how systems that utilize the *same* LLM in one or more of these roles should be described as having a Hybrid Functionality. These roles are described in detail below and summarized in Figure 1.

4.3.1 Generator. In this role, the LLM functions as the game content generator, where the outputs of the LLM are directly utilized as game content (with minimal to no post-processing). Direct influence on final content is critical to the classification of a Generator

Table 1: Preliminary classification of prior systems based on whether they qualify as PCG-LLM systems. A game content generation system is considered a PCG-LLM system if it contains at least one each of an LLM and a separate and distinct traditional PCG component, both of which contribute to the final content.

Paper Title	PCG-LLM System	Justification
MarioGPT (Sudhakaran et al., 2023)	No[1]	The LLM functions as a generator and is fine-tuned on level training data. No PCG components exist within the architecture.
Sokoban Generator (Todd et al., 2023)	No	The standalone LLM functions as a generator that is fine-tuned on level training data, no PCG components are present.
Metavoidal Generator (Nasir & Togelius, 2023)	No	LLM functions as a Generator, fine-tuned on level training data without intervention from PCG components.
Future Worlds (Kumaran et al., 2023)	Yes	The system has a clear LLM component (a natural language parser) and a PCG component (game level generator) that work sequentially to produce the content.
SCENECRAFT (Kumaran et al., 2023)	Yes	LLM and PCG components each contribute independently to the final playable scene by combining LLM-generated dialogue with algorithmic components that map emotions and gestures to game assets, translate outputs into a scripting language, and construct branching narrative graphs.
LLMGG (Hu et al., 2024)	Yes	Parser LLM used to interpret input produces VGDL rules that can be converted to game levels in a VGDL game engine such as GVGAI.
PANGeA (Buongiorno et al., 2024)	No	LLM is wrapped in a prompt management and validation framework with integration components that allow it to be accessed before and during gameplay. No PCG is involved.
Word2World (Nasir et al., 2024)	Yes	An LLM is involved in every step of world generation process and is paired with an evaluation loop combining traditional PCG methods such as A* pathfinding and LLM-based playability assessment to iteratively refine outputs. As a result, both LLM and PCG components influence the resultant content.
LLMaker (Gallotta et al., 2024)	Unclear[2]	LLM is used to process and interpret natural language inputs and generates parameters for backend functions. However, more information about the system is required to see if PCG components are present.
Agentic PCG (Jiang et al., 2026)	Yes	LLM agent and distinct PCG tool components, such as cellular automata, each contribute independently to the final level.

LLM. Furthermore, the LLM's outputs should *not* be utilized as a baseline for some downstream component to transform later on.

[1]MarioGPT with novelty search (NS-MarioGPT) would be considered a PCG-LLM system since novelty search is used to maximize the diversity of the LLMs outputs.
[2]LLMaker is treated as a near-miss case. The authors indicate that the LLM can access backend functions, but details about these functions are undisclosed. This highlights a limitation of the taxonomy: accurate classification depends on sufficient documentation. Systems that lack details run the risk of misclassification, reinforcing the recommendation that authors include system flowcharts and open source their code where possible.

The base generation pipeline for MarioGPT [22] and the Sokoban Generator by Todd et al. [24] present LLMs that are fine-tuned on level data operating as Generators. For MarioGPT, structured prompts are encoded and interpreted by the LLM before being converted to game levels. And in the case of the Sokoban generator, the model leans on the training data it has been exposed to to create playable, novel and diverse levels.

Hu et al. [4] present the LLMGG framework which uses an LLM functioning as a Generator by using the model to produce game rules in VGDL [20]. These rules are a codified video game which get

passed on to a VGDL-compatible game engine (such as GVGAI [17]) to render the game. LLMGG is a unique example of an LLM functioning on the edge of two roles. It almost operates as a Parser, but since its outputs are not passed on to any other components for manipulation and are directly rendered by a game engine, Generator becomes the dominant role for this LLM. SCENECRAFT by Kumaran et al. [8] demonstrates another application of a Generator LLM, utilizing one throughout each stage of its pipeline to ensure that narrative elements developed by algorithmic components are coherent with the emotional beats of the scene.

Systems that use generator LLMs may incorporate validation mechanisms to ensure the model's contents are valid, coherent and – where applicable – playable since satisfaction of these constraints are not guaranteed when LLMs directly output to the in-game representation. Procedures such as fine-tuning can support a model in producing reliable outputs, and techniques such as Retrieval-Augmented Generation (RAG) [9] can give a model a database to draw from to support its generation efforts. RAG-based approaches were absent from surveyed systems, representing an underexplored direction. Additionally, a generator could be paired with a downstream evaluation loop, or even an evaluator LLM to ensure that the resultant content is up to the designers standards before being utilized within a game.

4.3.2 Parser. LLMs assuming the role of a Parser indirectly influence the generation process by interpreting inputs and converting them from one modality into another for use by a downstream component. An LLM functioning as a Parser implies that it exists within either a multi-component architecture or that the model is functioning within multiple capacities, with parsing being one of them. Additionally, the resultant indirect influence on the parsed content is critical to classifying an LLM as a Parser, and for distinguishing a Parser LLM from a Generator LLM.

Example inputs to a Parser LLM could be natural language prompts or numerical codes that the system has been designed to recognize. Parser LLMs may, for example, convert these inputs into parameters for downstream components. Kumaran et al.'s *Future Worlds* exemplifies this as natural language descriptions are turned into game generator parameters [7].

Parser LLMs introduce considerations around hallucination and context window limitations. These risks can be mitigated through validation mechanisms such as a parameter checker that assesses a JSON file generated by the LLM against acceptable value ranges for downstream components. Context rot presents an additional concern for systems requiring sustained interaction, as degradation over extended prompting sequences could interfere with generation procedures or player experiences. This leaves the developer with a need to consider how their systems will be prompted and used.

4.3.3 Evaluator. An LLM functioning as an Evaluator may indirectly influence the content generation process by assessing how well the content satisfies criteria posed by the designer. The assessments of an evaluator are either reported out to the developer, or utilized within the system to tune the content. Word2World [14] is an example of an Evaluator LLM since it incorporates an LLM agent into its evaluation loop to assess level playability. A key criterion for classifying an LLM as an Evaluator is that the model does not manipulate the content directly. Even in the case of a Hybrid LLM – where an LLM switches between roles – the taxonomy assumes that the LLM is not manipulating the content while it is evaluating it. Should an LLM output manipulated content when prompted to evaluate it, then the taxonomy assumes that the LLM switches to the role of a Generator or a Parser, depending on the nature of the manipulation.

The use of Evaluator LLMs introduces the need for verifying that evaluations are done correctly and reliably across iterations. This need is amplified in systems where Evaluator outputs are fed back into the generation pipeline. LLM-based evaluation and validation has been explored in broader contexts [28]. As such, this taxonomy considers a validator LLM to be a mode of the Evaluator role where the LLM assesses content against correctness criteria rather than generating evaluation scores. PANGeA [1] presents an example of a validator LLM.

4.3.4 Orchestrator. LLM functioning as Orchestrators supervise other components within the system, providing multiple components with directives (either directly or indirectly) to influence the generation process with the ultimate goal of ensuring that content is aligned with the designer's specifications and expectations. A distinguishing feature of Orchestrator LLMs is that they cannot exist in isolation *and* produce output that is interpretable as content. The author hypothesizes that orchestrator LLMs should only be tasked with orchestration tasks to alleviate issues with LLM context windows and size constraints.

The idea of orchestration of game generation tasks has been explored before [10] but examples of orchestration are limited. Concurrent to the writing of this paper, Jiang et al. [5] presented a preprint of Agentic PCG – a framework that equips LLM agents with traditional PCG tools for editing and refining generated game levels across multiple games via the PCG Benchmark. The agents can respond to open-ended natural language instructions, while adhering to the constraints imposed by the different games. This work qualifies as an orchestration system as the model assumes a god-like perspective and discerns how to augment the game level with respect to the specified objectives by sending directives to the provided tools.

With increased interest in multi-agent frameworks and automatic game generation in recent years, it is likely that more systems that use LLMs to make design decisions for game content will arise. Generation pipelines that rely on orchestration hold a lot of promise for the exploration of multi-component coordination and autonomous game generation systems, but this area is currently underexplored. As a result, the design considerations and implications of Orchestrator LLMs is left open for future work.

4.3.5 Hybrid Functionality. Hybrid functionality (or a Hybrid LLM) is a proposed term for describing a single LLM component that is prompted to operate in more than one of the aforementioned roles. Systems with one or more LLMs fulfilling a single distinct role are not considered eligible for this category. A Hybrid LLM may arise when the same model is iteratively fed different prompts commanding it to operate in different roles, or when the model is fed a compound prompt asking it to serve in more than one role simultaneously.

The prevalence of Hybrid LLMs in surveyed systems is likely motivated by two properties of LLMs: their generalization capacity,

which enables a single model to perform meaningfully across multiple roles, and the computational and financial costs associated with deploying multiple distinct LLM components. Consolidating these roles into a single LLM component leverages the model's versatility while reducing overhead. As such, these categories are not intended to be rigid — hybrid functionality reflects the reality that LLMs are versatile enough to resist clean classification, and the taxonomy accommodates this.

Examples of systems with Hybrid LLMs include PANGeA [1] and Word2World [14], as the LLMs in those systems both operate as Generators, Parsers and Evaluators at different points in the system's execution cycle. LLMs functioning within a hybrid capacity introduce an increased risk of context window deterioration since the model must simultaneously retain information about multiple roles. This can compound hallucination and error rates. Surveyed Hybrid systems address these risks through validation systems and support structures to help the model stay on task and within the scope of the designer's intentions. As such, documenting each role a Hybrid LLM occupies is important for accurate classification and system comparison.

4.4 LLM Interaction Patterns Within PCG-LLM Systems

The taxonomy includes terms for capturing means that an LLM can communicate with other components within a system, to aid with the characterization of different architectures. Three of the four patterns are paired with alternate names – intended for synonymous use – that adopters can consider based on their preferences. These interaction patterns are described below and summarized in Figure 2.

4.4.1 Standalone. In this pattern, the LLM is the sole component of the system, therefore, there are no other components for the LLM to interact with. The LLM is responsible for receiving and processing inputs from the user without support from other components and there are no controllers or intermediary components attached to it. The only components attached to the LLM (if any) would be ones to render the LLM's outputs without altering the generated content beyond what is needed for utilization within the game. The Sokoban level generator by Todd et al. [24] is an example of this pattern as the model is not paired with any other manipulative components that assist it with the generation task. Surveyed work shows that resultant content is tightly coupled with model parameters, and in cases where models are fine-tuned, training data quality plays an important role in determining the quality of the content as well. No architectural mechanisms exist to assess quality before the model's own capabilities. Furthermore, context window limitations pose a compounding risk in standalone systems especially when repeated prompting occurs – which is permitted under this pattern. As such, the limitations of the LLM may become increasingly visible in the final output.

4.4.2 Sequential (or Pipelined). A sequential interaction pattern is distinguished by its unidirectional data flow through multiple components, resulting in a relatively flat architecture. This creates increased dependence on each component's outputs being correct. Without mechanisms for intervention, errors propagate forward, emphasizing that content quality must be considered at every stage of design. The unpredictability of LLMs also suggests that – although not required – LLM produced content is more likely to appear earlier in sequential pipelines, so their outputs can be corrected downstream before the final output. Additionally, it is possible to attach or embed other architectures into a sequential architecture. When this is done, indicating it clearly in system flowcharts and descriptions will support accurate classification. An example of this is Word2World [14] as evaluation is done within a looped architecture at the end of the sequential world generation pipeline. The generation pipeline in *Future Worlds* [7] also presents a clear example of a sequential architecture.

4.4.3 Bidirectional (or Collaborative). A bidirectional, or collaborative interaction pattern characterizes a two-way dialogue between system components. Two modes of operation for the bidirectional interaction pattern emerged from the surveyed literature. The first mode is routine access, where a system periodically calls on a bidirectional component which suggests suitability for continuous or in-game content generation. The second is criteria-driven communication – where the LLM and a collaborator component iterate upon the content until a condition (or set of conditions) are met. One component produces the content, and the other provides feedback until the stop condition is met. In criteria-driven communication, evaluation metrics become the controller of the output's timing and quality as one of the components must deem the content as sufficient before releasing it to downstream components. If evaluation of the content is handled by a tertiary evaluator, a Bidirectional system becomes a Looped one. The iterative nature of bidirectional interactions can make identifying which component is responsible for the final output's quality more difficult to determine. The level generation in Metavoidal [15] is a solid example of bidirectional communication, both between an LLM and a human, and then again with an LLM and an automated evaluation component. PANGeA [1] exemplifies routine access – the other mode for this pattern – as the system calls on the LLM during gameplay to generate narrative content on demand.

4.4.4 Looped (or Within A Loop). A Looped interaction pattern requires at least three system components with at least one being an LLM. In the surveyed literature, Looped patterns were observed exclusively in evaluation contexts, suggesting this pattern is particularly well suited for quality assurance tasks within a generation pipeline and iterative content design processes. A Looped system can range from very simple structures to complex architectures. The complexity of looped architectures introduces risk of error propagation compounding across components within the loop and as iterations of the loop increase. This suggests that careful coordination of LLM and PCG components is a key design consideration for this architecture. This pattern is underexplored within the surveyed corpus, and additional community engagement would help surface further design implications. The evaluation loop in Word2World [14] is an example of a looped interaction pattern.

Agentic PCG [5] demonstrates a sequential pipeline in which the generation stage itself embodies a looped pattern where the agent iteratively calls tools and receives evaluation feedback to refine game levels before producing a final output. The tool calling and optimization components within the framework's generation loop

prevent this interaction pattern from being purely bidirectional. Orchestration is still emerging as a role, though agentic workflows naturally lend themselves to looped interaction patterns [5, 10]. The nesting of interaction patterns as systems increase in complexity is something surveyed systems suggest, but this exploration is left for future work.

5 Application of Taxonomy

This section demonstrates how to utilize this taxonomy by applying it to past works. Generally, when considering a work, adopters of this taxonomy should read through the work and extract details about the system's architecture. From there, every component that contains an LLM should be identified, as well as the components that interact with it. The functionality of the LLMs in each component and the ones around it should then be used to determine the role of the LLM and the model-component interaction patterns demonstrated within the system.

5.1 PANGeA

PANGeA, developed by Buongiorno et al. [1], is a game engine plugin that enables dynamic narratives by allowing players to engage with NPCs through free-form text. Structured natural language input from the user is converted to JSON and handed off to a server-side architecture with two components relevant to this taxonomy: an LLM interface that facilitates communication with an external LLM, and a validation system that checks whether the model's outputs align with the game's rules.

PANGeA does not qualify as a PCG-LLM system, as no component uses traditional PCG approaches to shape the game's contents. The LLM exhibits hybrid functionality, operating as a Generator by producing in-game content, a Parser by interpreting structured JSON inputs before generation, and an Evaluator in the mode of a validator by assessing content against the game's rules rather than producing evaluation scores. The LLM and the server-side interface communicate bidirectionally, allowing the model to be invoked repeatedly during initialization or during gameplay. This pattern shifts the LLM from a participant in a sequence to an on-demand resource that can directly shape the player's experience.

5.2 Future Worlds

Future Worlds is a 3D virtual game developed by Kumaran et al. [7] that teaches players about sustainability by showing how their decisions affect resources like water and energy. The system uses an LLM to extract design targets and parameters for a procedural level generator, and exemplifies a sequential generation pipeline: designer input primes the LLM to interpret user prompts and convert them into constraints for the level generator. PCG techniques then produce candidate levels, which are assessed by a deep reinforcement learning evaluator before being passed to the playable level generator in Unity. The natural language LLM functions as a Parser, extracting relevant information from the prompt and outputting constraints in a structured form – a 4-tuple in this case – for the downstream generator. Because data flows in one direction with no return path to the model, the LLM participates within a Sequential interaction pattern.

6 Discussion & Limitations

Research at the intersection of game content generation and LLMs has given rise to diverse systems with unique architectures, making the need for common terminology clear. RQ1 and RQ2 are addressed in the taxonomy presented in Section 4 and validated in Section 5. RQ3 is addressed through the coverage of the design implications of each LLM role and interaction pattern in Section 4.

The classification of the systems in the corpus is presented in Table 1 and Table 2, classifying PCG-LLM systems and LLM roles and interaction patterns respectively. The data shows that of the nine works, four of them were PCG-LLM systems, four of them were not, and one of them was unclear. This indicates that the definition of a PCG-LLM system provides a meaningful distinction within the field, and that appropriate classification is highly dependent on the amount of information that authors provide about their systems.

As for LLM roles, the Generator role is the most common and most frequently occurring primary LLM role, appearing in seven of nine surveyed works. This indicates that researchers have focused on the generative capacities of LLMs in their applications. Parser is the most common secondary role, appearing in four of nine works, indicating that researchers recognize the information extraction capabilities of LLMs as viable supports for generation tasks. Evaluator is the third most listed LLM role, sitting in the tertiary position for two of the nine works. This implies that LLMs as evaluators is a recognized capacity, but it has yet to be rigorously explored in the context of game content generation. The Orchestrator role was absent from the primary 2022-2024 corpus, though concurrent work such as Agentic PCG suggests this pattern is beginning to emerge. The Sequential interaction pattern was the most common, present in six of the nine works. The simplicity and creative control allowed by a Sequential pattern likely contributes to its prominence as it allows the designer to build a system that can create, refine, and correct information in a straightforward fashion. Bidirectional was the second most prominent interaction pattern, appearing in two of the nine works, the Metavoidal level generator and PANGeA. The former required the LLM to iterate on level content, and the latter required routine access to the LLM to generate content when necessary. Only the Sokoban generator by Todd et al. [24] was classified as a Standalone system. The Looped architecture only appeared once in Word2World and was attached to a sequential architecture in an evaluation capacity. These results suggest that most research has focused on how LLMs can contribute to more complex architectures, with limited exploration into other patterns, leaving room for more areas of study.

The strengths of the taxonomy include its ability to provide a more reliable common ground to use when comparing systems. PCG is subjective and LLMs are black-box models, but software architecture and functionality are more directly observable than model internals or content quality alone. This opens the door to several comparative research questions that were previously difficult to ask such as: "Does an architecture with an LLM functioning as a Parser create more controllable content than an architecture that utilizes an LLM as a Generator?" These terms were designed to be scalable and can be applied prescriptively and descriptively to help practitioners reason about their design decisions before expending resources.

Table 2: Classification of LLM roles and interaction patterns across prior systems. Primary roles denote the dominant function of the LLM within the pipeline, while secondary roles capture additional responsibilities. Interaction patterns describe how LLM components interface with other modules or processes.

Paper Title	Primary LLM Role	Other LLM Roles	LLM Interaction Patterns
MarioGPT (Sudhakaran et al., 2023)	Generator	N/A	Sequential
Sokoban Generator (Todd et al., 2023)	Generator	N/A	Standalone
Metavoidal Generator (Nasir & Togelius, 2023)	Generator	N/A	Bidirectional
Future Worlds (Kumaran et al., 2023)	Parser	N/A	Sequential
SCENECRAFT (Kumaran et al., 2023)	Generator	Parser	Sequential
LLMGG (Hu et al., 2024)	Generator	Parser	Sequential
PANGeA (Buongiorno et al., 2024)	Generator	Parser, Evaluator	Bidirectional
Word2World (Nasir et al., 2024)	Generator	Parser, Evaluator	Sequential, Looped
LLMaker (Gallotta et al., 2024)	Parser	N/A	Sequential
Agentic PCG (Jiang et al., 2026)	Orchestrator	N/A	Sequential, Looped

This taxonomy has yet to be rigorously applied to a significant sample of eligible systems and classifications are limited by the availability of architectural detail. Systems with incomplete documentation risk miscategorization and some systems may sit at the boundary of the PCG-LLM system definition where insufficient detail renders assessments inconclusive. This taxonomy is scoped to works published between 2022 and 2024, reflecting the surge in research that followed the release of early LLM-based level generators on arXiv. Works beyond this window were not considered and it is possible that those works may surface categories or edge cases not represented by the surveyed corpus. Finally, all terminology and classifications reflect the assessments of a single author, introducing subjectivity that community validation will be essential to address. As such, the author presents this as a proposed framework and welcomes critique, application and adaptation from the broader community.

Researchers adopting this taxonomy would benefit from including system flowcharts in their work. The majority of surveyed works (7 out of 9) did so, which meaningfully aided classification efforts by clarifying how LLM-powered components interact. Standardizing this practice would lower barriers around system comprehension, aid comparisons between systems, and support entry for prospective researchers. Additionally, although not uncommon, the practice of open-sourcing code should be further promoted to increase community engagement. Releasing reproducible code, system prompts, the exact models used (along with details about their hyperparameters and how to run them) is also essential for the field's advancement.

7 Conclusion & Future Work

This work introduced PCG-LLM systems as a term for content generation architectures that incorporate both PCG and LLM components, and proposed a two-dimensional taxonomy characterizing the functional roles LLMs assume and the interaction patterns through which they participate in these systems. A single LLM can also function in multiple roles, which the taxonomy recognizes as a Hybrid Functionality.

The taxonomy was used to classify nine works at the intersection of PCG and LLMs with eligible systems. The results suggest that PCG-LLM systems are a meaningful distinction, and identified trends in LLM roles and interaction patterns. LLMs that function as Generators are the most common, followed by LLMs that function as Parsers and Evaluators. This shows that most research has focused on the generative capacities of LLM for game content, with other roles and hybrid interactions remaining unexplored. LLMs that orchestrate – or supervise – design processes are an emergent pattern with a lot of promise for future research. Sequential interaction patterns were the most prominent, likely due to the approachability they provide to designers who want to use LLMs in content generation processes. The design implications of each LLM role and interaction pattern were also described.

Future work includes expanding the corpus beyond the 2022-2024 window – particularly as frameworks like Agentic PCG [5] begin to instantiate the Orchestrator role this taxonomy anticipated – and investigating relationships between architectural patterns and evaluation outcomes. Reporting standards for game content generation systems can also be advanced in part by the terms proposed by this work, leading to standardized practices for visualizing system components and detailing how each one does or does not contribute to the final content.

This taxonomy is a starting point, not a final word. Its value will be determined by how the community engages with it through adoption, critique, and refinement as the field continues to define what it means to generate game content with LLMs and PCG approaches.

Acknowledgments

The author thanks their colleagues who provided support, feedback and guidance on this work.

References

[1] Steph Buongiorno, Lawrence Klinkert, Zixin Zhuang, Tanishq Chawla, and Corey Clark. 2024. PANGeA: procedural artificial narrative using generative AI for turn-based, role-playing video games. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, Vol. 20. 156–166.

[2] Roberto Gallotta, Antonios Liapis, and Georgios Yannakakis. 2024. LLMaker: A game level design interface using (only) natural language. In *2024 IEEE Conference on Games (CoG)*. IEEE, 1–2.

[3] Roberto Gallotta, Graham Todd, Marvin Zammit, Sam Earle, Antonios Liapis, Julian Togelius, and Georgios N Yannakakis. 2024. Large language models and games: A survey and roadmap. *IEEE Transactions on Games* (2024).

[4] Chengpeng Hu, Yunlong Zhao, and Jialin Liu. 2024. Game generation via large language models. In *2024 IEEE Conference on Games (CoG)*. IEEE, 1–4.

[5] Zehua Jiang, Sam Earle, Ahmed Khalifa, and Julian Togelius. 2026. Agentic PCG: Procedural Content Generation via Tool-using LLMs. *Available at SSRN 6499021* (2026).

[6] Ahmed Khalifa, Roberto Gallotta, Matthew Barthet, Antonios Liapis, Julian Togelius, and Georgios N. Yannakakis. 2025. The Procedural Content Generation Benchmark: An Open-source Testbed for Generative Challenges in Games. In *Proceedings of the 20th International Conference on the Foundations of Digital Games (FDG '25)*. Association for Computing Machinery, New York, NY, USA, Article 27, 12 pages. doi:10.1145/3723498.3723794

[7] Vikram Kumaran, Dan Carpenter, Jonathan Rowe, Bradford Mott, and James Lester. 2023. End-to-end procedural level generation in educational games with natural language instruction. In *2023 IEEE Conference on Games (CoG)*. IEEE, 1–8.

[8] Vikram Kumaran, Jonathan Rowe, Bradford Mott, and James Lester. 2023. Scenecraft: automating interactive narrative scene generation in digital games with large language models. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, Vol. 19. 86–96.

[9] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in neural information processing systems* 33 (2020), 9459–9474.

[10] Antonios Liapis, Georgios N Yannakakis, Mark J Nelson, Mike Preuss, and Rafael Bidarra. 2018. Orchestrating game generation. *IEEE Transactions on Games* 11, 1 (2018), 48–68.

[11] Jialin Liu, Sam Snodgrass, Ahmed Khalifa, Sebastian Risi, Georgios N Yannakakis, and Julian Togelius. 2021. Deep learning for procedural content generation. *Neural Computing and Applications* 33, 1 (2021), 19–37.

[12] Mahdi Farrokhi Maleki and Richard Zhao. 2024. Procedural content generation in games: A survey with insights on emerging llm integration. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, Vol. 20. 167–178.

[13] Xinyu Mao, Wanli Yu, Kazunori D Yamada, and Michael R. Zielewski. 2024. Procedural Content Generation via Generative Artificial Intelligence. arXiv:2407.09013 [cs.AI] https://arxiv.org/abs/2407.09013

[14] Muhammad U Nasir, Steven James, and Julian Togelius. 2024. Word2world: Generating stories and worlds through large language models. *arXiv preprint arXiv:2405.06686* (2024).

[15] Muhammad U Nasir and Julian Togelius. 2023. Practical PCG Through Large Language Models. arXiv:2305.18243 [cs.CL] https://arxiv.org/abs/2305.18243

[16] Nintendo. 1985. Super Mario Bros. Game. www.nintendo.com Platform: Nintendo Entertainment System (NES), Publisher: Nintendo, Released: September 13, 1985 (JP).

[17] Diego Perez-Liebana, Jialin Liu, Ahmed Khalifa, Raluca D. Gaina, Julian Togelius, and Simon M. Lucas. 2019. General Video Game AI: A Multitrack Framework for Evaluating Agents, Games, and Content Generation Algorithms. *IEEE Transactions on Games* 11, 3 (2019), 195–214. doi:10.1109/TG.2019.2901021

[18] Ken Perlin. 1985. An image synthesizer. *ACM Siggraph Computer Graphics* 19, 3 (1985), 287–296.

[19] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. 2019. Language models are unsupervised multitask learners. *OpenAI blog* 1, 8 (2019), 9.

[20] Tom Schaul. 2013. A video game description language for model-based or interactive learning. In *2013 IEEE Conference on Computational Inteligence in Games (CIG)*. IEEE, 1–8.

[21] Noor Shaker, Julian Togelius, and Mark J. Nelson. 2016. *Procedural Content Generation in Games: A Textbook and an Overview of Current Research*. Springer. doi:10.1007/978-3-319-42716-4

[22] Shyam Sudhakaran, Miguel González-Duque, Matthias Freiberger, Claire Glanois, Elias Najarro, and Sebastian Risi. 2023. Mariogpt: Open-ended text2level generation through large language models. *Advances in Neural Information Processing Systems* 36 (2023), 54213–54227.

[23] Adam Summerville, Sam Snodgrass, Matthew Guzdial, Christoffer Holmgård, Amy K Hoover, Aaron Isaksen, Andy Nealen, and Julian Togelius. 2018. Procedural content generation via machine learning (PCGML). *IEEE Transactions on Games* 10, 3 (2018), 257–270.

[24] Graham Todd, Sam Earle, Muhammad Umair Nasir, Michael Cerny Green, and Julian Togelius. 2023. Level generation through large language models. In *Proceedings of the 18th International Conference on the Foundations of Digital Games*. 1–8.

[25] Julian Togelius, Emil Kastbjerg, David Schedl, and Georgios N. Yannakakis. 2011. What is procedural content generation? Mario on the borderline. In *Proceedings of the 2nd International Workshop on Procedural Content Generation in Games* (Bordeaux, France) *(PCGames '11)*. Association for Computing Machinery, New York, NY, USA, Article 3, 6 pages. doi:10.1145/2000919.2000922

[26] Julian Togelius, Georgios N. Yannakakis, Kenneth O. Stanley, and Cameron Browne. 2011. Search-Based Procedural Content Generation: A Taxonomy and Survey. *IEEE Transactions on Computational Intelligence and AI in Games* 3, 3 (2011), 172–186. doi:10.1109/TCIAIG.2011.2148116

[27] Georgios N. Yannakakis and Julian Togelius. 2025. *Artificial Intelligence and Games*. Springer Nature. https://gameaibook.org.

[28] Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. 2023. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in neural information processing systems* 36 (2023), 46595–46623.